

FIG. 1

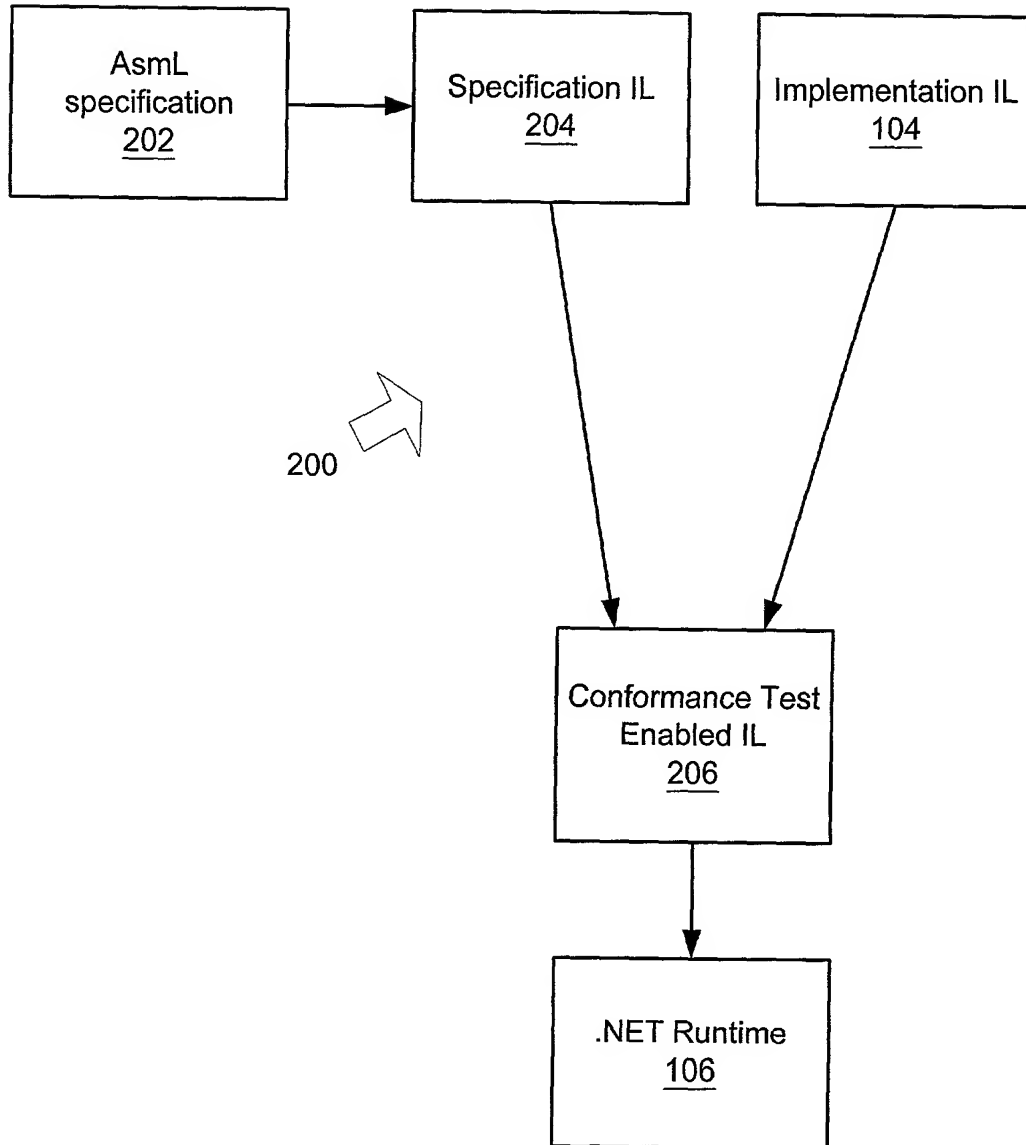



FIG. 2

```
302 class Hashtable$Contract extends Hashtable
304   constraint
306     let indices = {0..keys.Length - 1}
308     keys != null and values != null and
310     keys.Length = values.Length and
312     (forall i in indices holds keys[i ] != null ) and
314     (forall i , j in indices holds keys[i ] = keys[j ] => i = j )
318   Object set (Object key, Object value)
319   ensure
320     (key = null and thrown is ArgumentNullException) or
322     (exists i in {0..keys'.Length - 1} holds keys[i ]' = key and
324     values[i ]' = value and
326     result = value)
```



300

FIG. 3


```
402 class Hashtable$Checked : IDictionary {
404     void Hashtable$Invariant() {
405         ASSERT([[constraint clause from Fig. 3]]);
406     }
407     void set$Pre (Object key, Object val) { ASSERT(true); }
408     void set$Post (Object key, Object value, Object result) {
410         ASSERT([[ensure clause from Fig. 3]]);
412     }
414     Object[ ] keys; Object[ ] values;
418     Object set (Object key, Object value) {
420         Object result ;
422         Hashtable$Invariant();
424         set$Pre(key, value);
426         try {
428             [[body of the set method from the implementation code]]
432             <return value / result = value; break END;>
434         } catch (Exception e) {
436             result = e;
438         }
440     END :
444         Hashtable$Invariant();
446         set$Post(key, value, result );
448         if (result is Exception) throw result ; else return result ;
450     }}
```



400

FIG. 4


502 class IDictionary\$Contract implements IDictionary
504 var map as Map of Object to Object
506 constraint null notin domain(map)
510 Object set (Object key, Object value)
511 ensure
512 (key = null and thrown is ArgumentNullException) or
514 (map(key)' = value and result = value)



500


FIG. 5

604 Hashtable::abstraction () as IDictionary\$Contract {
606 return new IDictionary\$Contract(
608 {keys(i) |-> values(i) | i in {0..keys.Length - 1} where keys(i) != null}); }



600


FIG. 6

700


```
702 class IDictionary$Contract implements IDictionary
704   var map as Map of Object to Object
706   var enums as Set of IEnumerator
708   constraint null notin domain(map) and null notin enums
712   Object set (Object key, Object value)
714     step if key = null
716       throw new ArgumentNullException()
718       map(key) := value
720   step forall e in enums
722     e.Invalidate()
724   step return map(key)
```

FIG. 7


800



```
802 class IDictionary$Contract {  
804     AsmL.Map map = new AsmL.Map();  
806     AsmL.Set enums = new AsmL.Set();  
808     void set$Pre (Object key, Object val) { ASSERT(true); }  
810     void set$Post (Object key, Object val, Object result) {  
812         try {  
814             [[body of the specification of the the set method of Fig. 7]]  
815             <return e / ASSERT(result == e); return; >  
816         } catch (Exception e) {  
818             ASSERT(result.GetType() == e.GetType()); } } }
```

FIG. 8


```
902 class Hashtable$Checked : IDictionary {
904   IDictionary$Contract contract = new IDictionary$Contract();
906   Object[] keys; Object values[];
908   Object set(Object key, Object value) {
910     Object result;
912     contract.Invariant();
914     contract.set$Pre(key,value);
916     try {
918       [[body of the set method from the implementation code]]
920       <return e / result = e; break END; >
922     } catch (Exception ex) { result = ex;}
924     END:
926     contract.set$Post(key,value,result);
928     contract.Invariant();
930     if ( result is Exception ) throw result; else return result; }}
```



900

FIG. 9

```
1002 class Hashtable$Checked : IDictionary {
1004   IDictionary$Contract contract;
1006   IDictionary$Contract initialization() { . . . }
1008   Hashtable(. . .) {
1009     //initialize implementation variables here...
1010     contract = initialization(); }}
```



1000

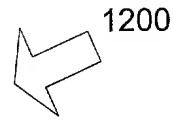
FIG. 10


```
1102 set (Object key, Object value)
1104   if key = null
1106       throw ArgumentNullException
1108   map(key) := value
1110   if key in domain(map)
1112       choose r in { map(key), value }
1114       return r
1115   ifnone
1116       throw new RuntimeException()
1116   else
1118       return value
```



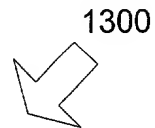
1100

FIG. 11



```
1202 class IDictionary$Contract {  
1204   AsmL.Map map = new AsmL.Map();  
1206   AsmL.Set enums = new AsmL.Set();  
1208   IDictionary$Contract$Set setInstance;  
1210   void set$Pre(Object key, Object value) {  
1212     setInstance = new IDictionary$Contract$Set(key,value);  
1214     ASSERT(true);  
1216     setInstance.Step(); }  
1220   void set$Post(Object key, Object value, Object result) {  
1222     setInstance.impl_result = result;  
1224     ASSERT(true);  
1226     setInstance.pc = END;  
1228     setInstance.Step(); }
```

FIG. 12



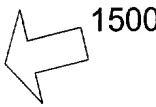
```
1232 class IDictionary$Contract$Set : ISteppable, IDictionary$Contract {  
1234     Object key;  
1236     Object value;  
1238     Object result;  
1240     Object impl_result;  
1242     int pc;  
1244     void Step();  
1246     IDictionary$Contract$Set(Object k, Object v) {  
1248         key = k; value = v; pc = 0; }  
1249 }
```

FIG. 13

1400
1250 void IDictionary\$Contract\$Set::Step() {
1252 while (true) {
1254 switch (pc) {
1256 case 0: Stack.Push(new EmptyFrame()); pc += 1; break;
1258 case 1: try {
1260 if (key == null)
1262 throw new ArgumentNullException();
1264 map(key) = value;
1266 if (Stack.NoOfCalls() > 0) {
1268 pc += 1; return; }
1270 pc += 3; break;
1272 } catch (Exception ex) {
1274 result = ex; pc = END; return; }
1276 case 2: ASSERT([constraint from Figure 7] && true); pc += 1; return;
1278 case 3: if (Stack.NoOfCalls() > 0) { pc -= 1; return; }
1280 pc += 1; break;
1280 }

FIG. 14

1282 case 4: try {
1286 foreach(IEnumerator e; enums)
1288 Stack.Add(new Call(this,e,Invalidate,null))
1290 [[repeat lines 1266-1274]]
1292 case 5: ASSERT([[constraint from Figure 7]] && map(key) == value);
1294 pc += 1; return;
1296 case 6: [[repeat lines 1278-1280]]



1500

FIG. 15

1298 case 7: try { result = map(key);
1300 if (Stack.NoOfCalls() > 0) {pc += 1; return; }
1302 pc = END; return;
1304 } catch (Exception ex) {
1306 result = ex; pc = END; return; }
1308 case 8:
1310 ASSERT([constraint from Figure 7] && map(key) == value);
1312 pc += 1; return;
1314 case 9: if (Stack.NoOfCalls() > 0) {pc -= 1; return; }
1316 pc = END; return;
1318 case END:
1320 ASSERT(Stack.NoOfCalls() == 0);
1322 ASSERT(result is Exception ?
1324 result.GetType() == impl_result.GetType() :
1326 result == impl_result);
1328 Stack.Pop();
1330 return; } } }

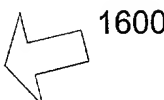
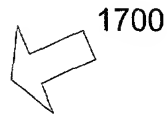


FIG. 16



```
1350 Invalidate() {  
1352   Object result;  
1354   Call c = Stack.SelectCall(this, "Invalidate", null);  
1356   if (c != null) c.Caller.Step();  
1358   try { valid := false } catch (Exception ex) { result = ex; }  
1362   if (c != null) { Stack.Remove(c); c.Caller.Step(); }  
1364   if ( result is Exception ) throw result; else return; }
```

FIG. 17

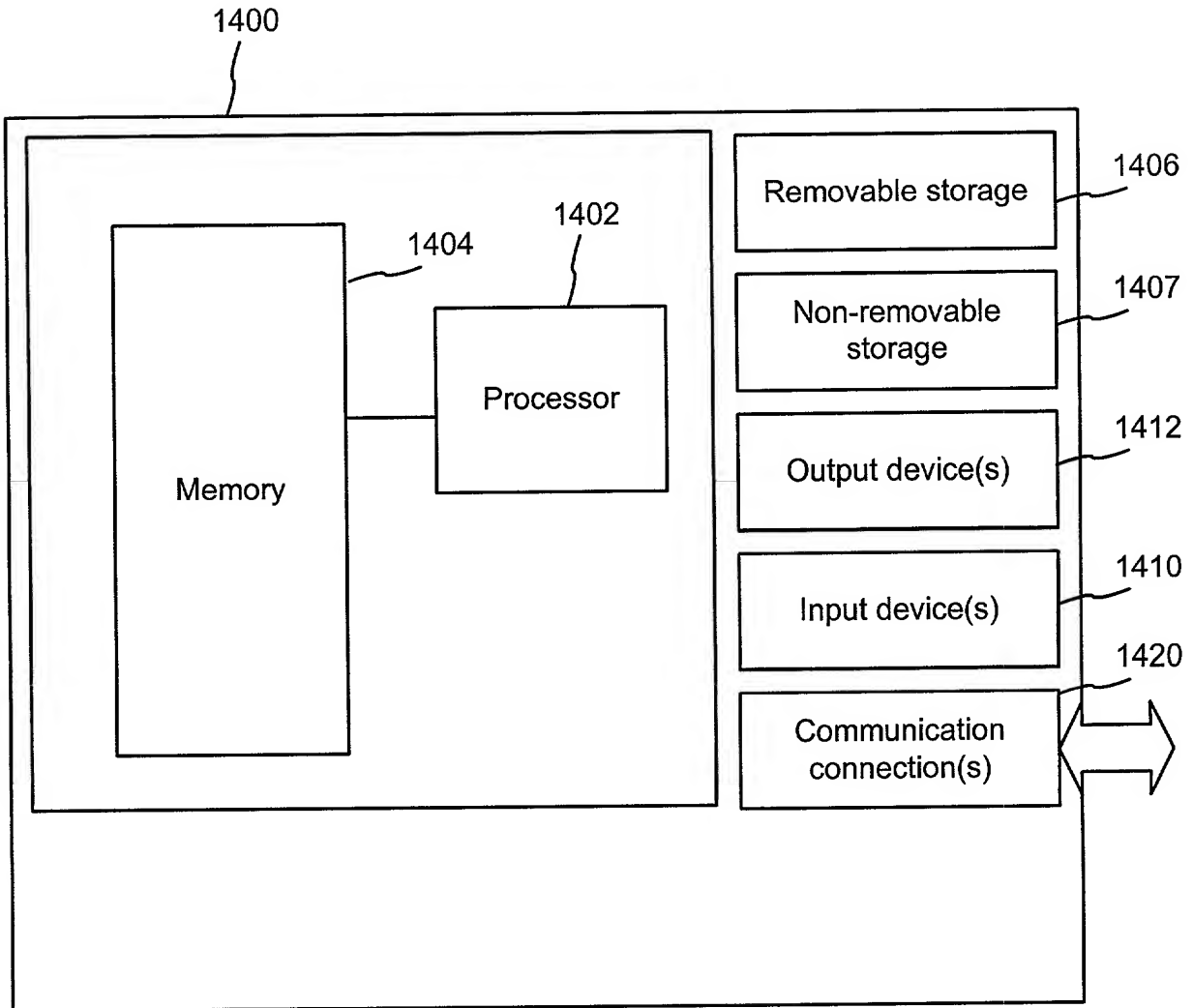


FIG. 18